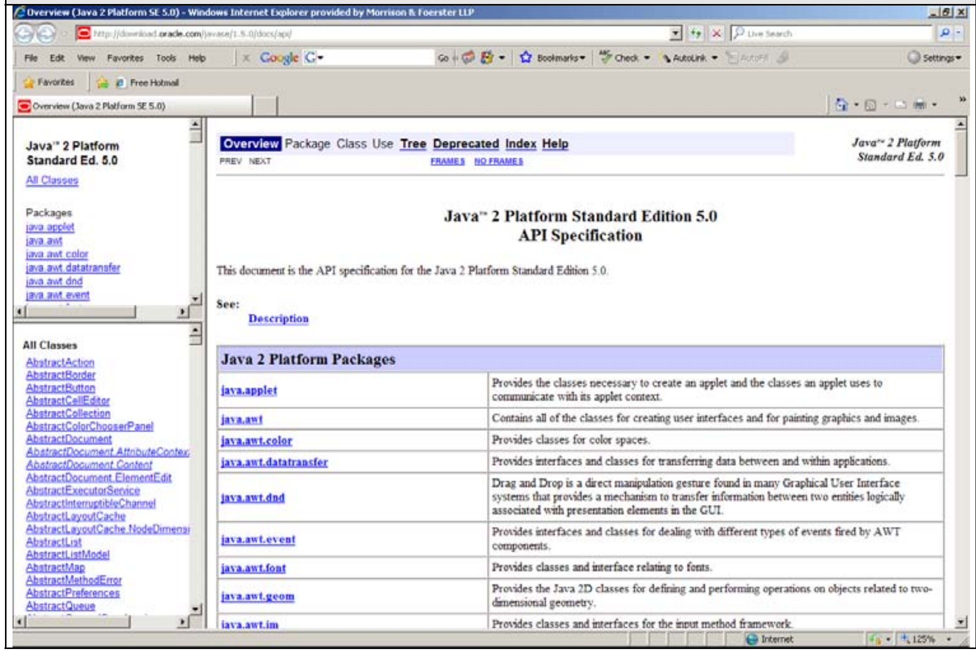
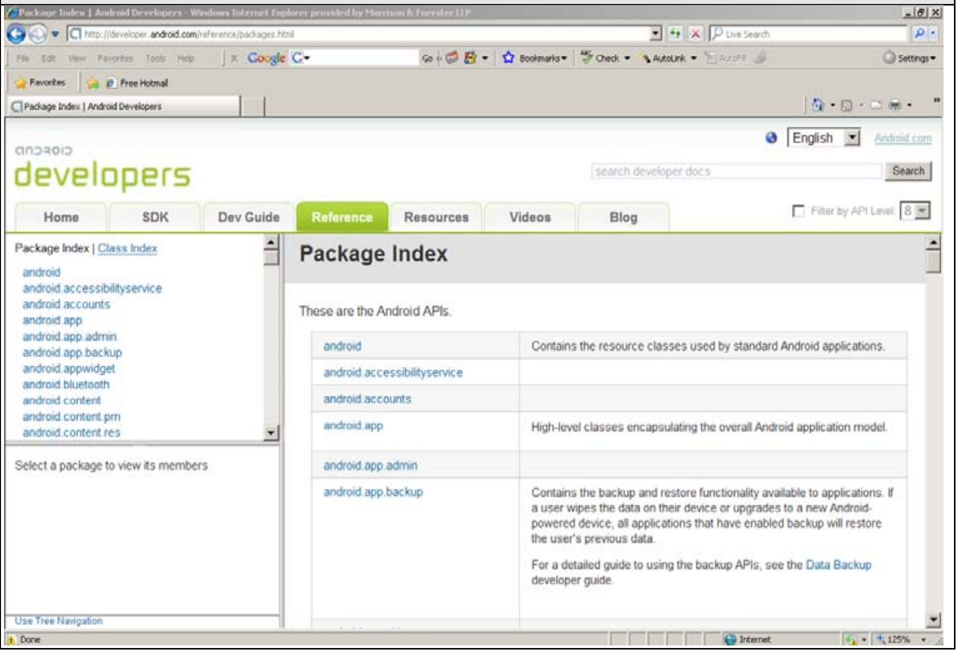
EXHIBIT D

Exhibit A

Java^{TM} 2 Platform Standard Edition 5.0 API Specification	Android APIs
	

Overview (Java 2 Platform SE 5.0) - Windows Internet Explorer provided by Morrison & Foerster LLP
http://download.oracle.com/javase/1.5.0/docs/api/
Live Search
File Edit View Favorites Tools Help
Google
Go
Bookmarks
Check
AutoLink
AutoFill
Settings
Favorites
Free Hotmail
Overview (Java 2 Platform SE 5.0)
Internet
125%

Java™ 2 Platform
Standard Ed. 5.0
All Classes

Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

All Classes
AbstractAction
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContex
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensi
AbstractList
AbstractListModel
AbstractMap
AbstractMethodError
AbstractPreferences
AbstractQueue

Overview Package Class Use **Tree** **Deprecated** **Index** **Help**
PREV NEXT FRAMES NO FRAMES

Java™ 2 Platform
Standard Ed. 5.0

Java™ 2 Platform Standard Edition 5.0
API Specification

This document is the API specification for the Java 2 Platform Standard Edition 5.0.

See:
Description

Java 2 Platform Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.

Package Index | Class Index

android
android.accessibilityservice
android.accounts
android.app
android.app.admin
android.app.backup
android.appwidget
android.bluetooth
android.content
android.content.pm
android.content.res

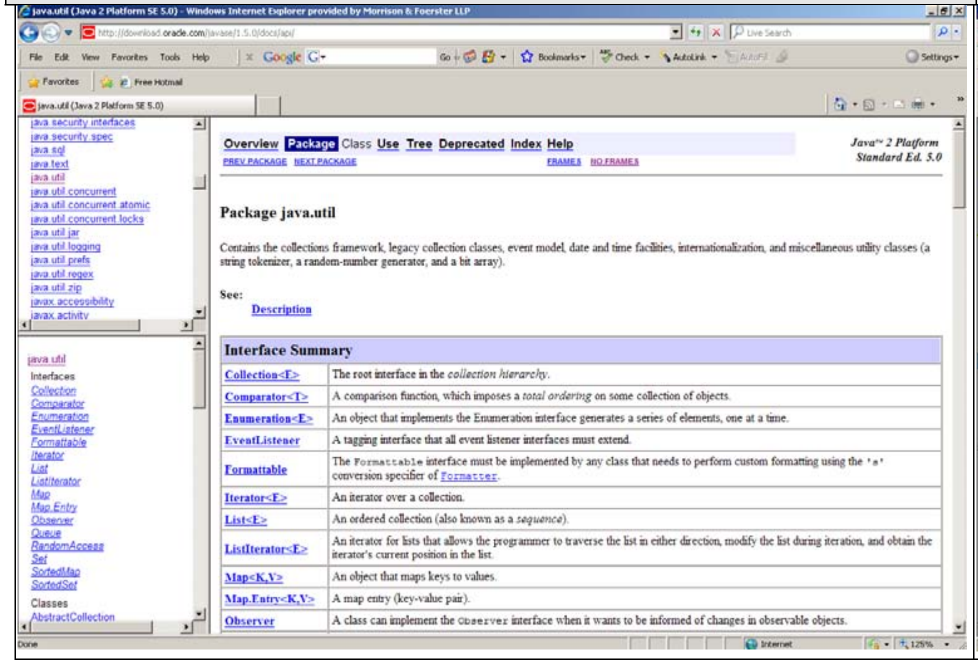
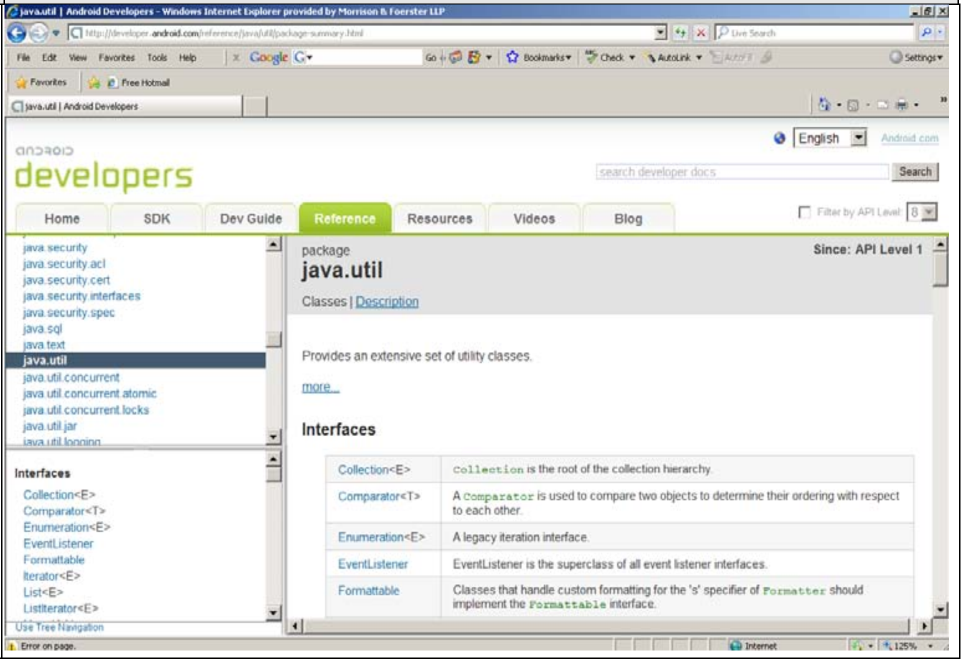
Select a package to view its members

Package Index

These are the Android APIs.

android	Contains the resource classes used by standard Android applications.
android.accessibilityservice	
android.accounts	
android.app	High-level classes encapsulating the overall Android application model.
android.app.admin	
android.app.backup	Contains the backup and restore functionality available to applications. If a user wipes the data on their device or upgrades to a new Android-powered device, all applications that have enabled backup will restore the user's previous data. For a detailed guide to using the backup APIs, see the Data Backup developer guide.

Exhibit A

Java^{TM} 2 Platform Standard Edition 5.0 API Specification	Android APIs
	

java.util (Java 2 Platform SE 5.0) - Windows Internet Explorer provided by Morrison & Foerster LLP

java.security.interfaces
java.security.spec
java.sql
java.text
java.util
java.util.concurrent
java.util.concurrent.atomic
java.util.concurrent.locks
java.util.jar
java.util.logging
java.util.prefs
java.util.regex
java.util.zip
javax.accessibility
javax.activity

java.util
Interfaces
Collection
Comparator
Enumeration
EventListener
Formattable
Iterator
List
ListIterator
Map
Map.Entry
Observer
Queue
RandomAccess
Set
SortedMap
SortedSet
Classes
AbstractCollection

Overview **Package** Class **Use** **Tree** **Deprecated** **Index** **Help**
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Java™ 2 Platform Standard Ed. 5.0

Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

See:
Description

Interface Summary	
Collection<E>	The root interface in the *collection hierarchy*.
Comparator<T>	A comparison function, which imposes a *total ordering* on some collection of objects.
Enumeration<E>	An object that implements the Enumeration interface generates a series of elements, one at a time.
EventListener	A tagging interface that all event listener interfaces must extend.
Formattable	The `Formattable` interface must be implemented by any class that needs to perform custom formatting using the `'s'` conversion specifier of `Formatter`.
Iterator<E>	An iterator over a collection.
List<E>	An ordered collection (also known as a *sequence*).
ListIterator<E>	An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.
Map<K,V>	An object that maps keys to values.
Map.Entry<K,V>	A map entry (key-value pair).
Observer	A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

java.util | Android Developers - Windows Internet Explorer provided by Morrison & Foerster LLP

java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.util
java.util.concurrent
java.util.concurrent.atomic
java.util.concurrent.locks
java.util.jar
java.util.logging

Interfaces
Collection<E>
Comparator<T>
Enumeration<E>
EventListener
Formattable
Iterator<E>
List<E>
ListIterator<E>

Use Tree Navigation

Error on page.

package
java.util

Since: API Level 1

Classes | Description

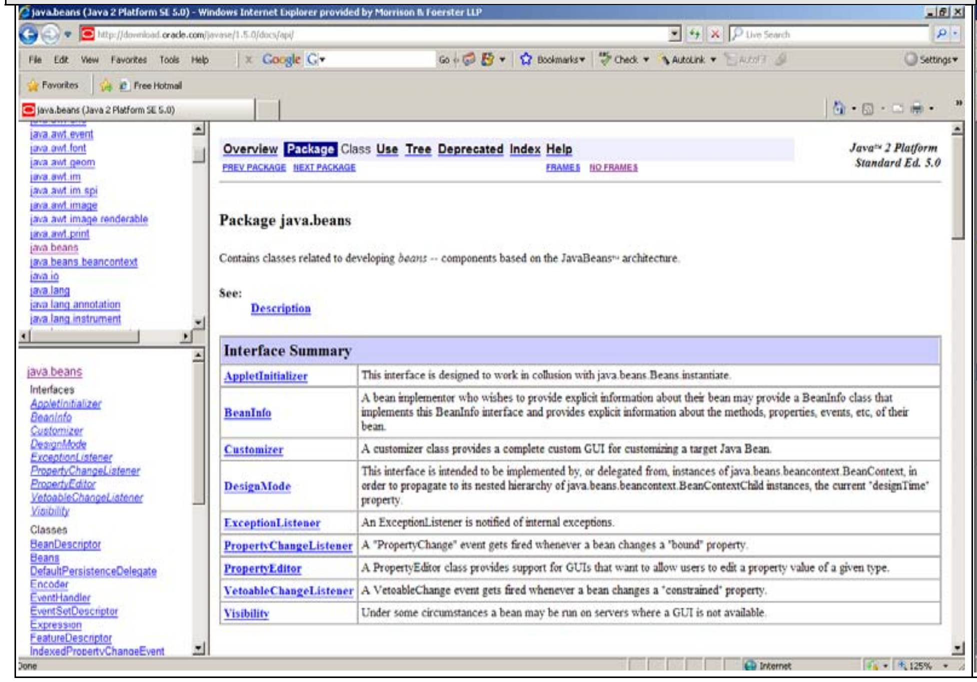
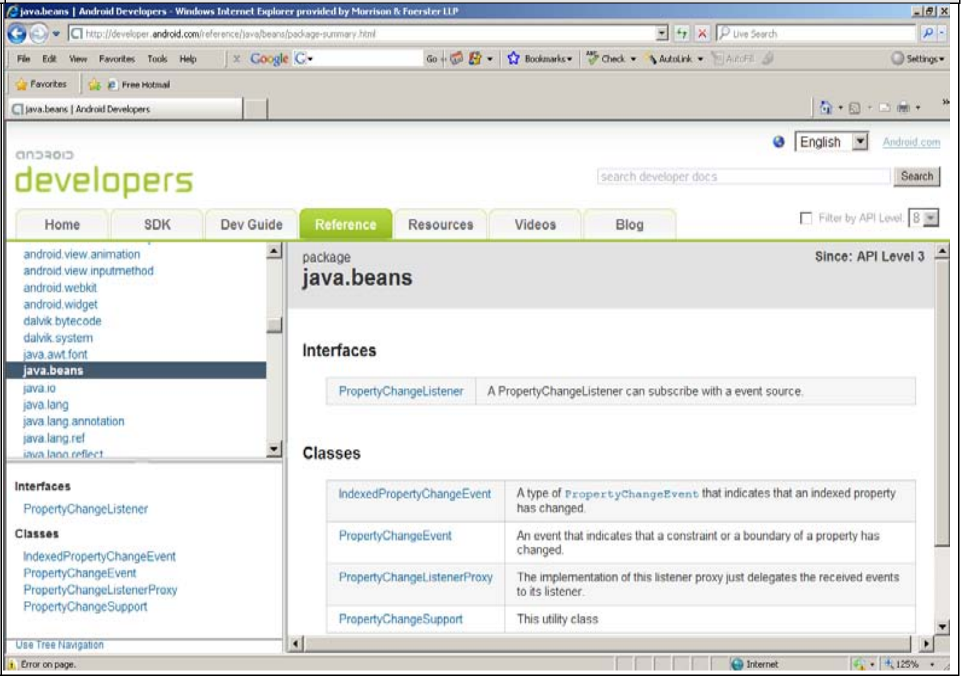
Provides an extensive set of utility classes.

more...

Interfaces

Collection<E>	`Collection` is the root of the collection hierarchy.
Comparator<T>	A `Comparator` is used to compare two objects to determine their ordering with respect to each other.
Enumeration<E>	A legacy iteration interface.
EventListener	EventListener is the superclass of all event listener interfaces.
Formattable	Classes that handle custom formatting for the 's' specifier of `Formatter` should implement the `Formattable` interface.

Exhibit A

Java^{TM} 2 Platform Standard Edition 5.0 API Specification	Android APIs
	

java.beans (Java 2 Platform SE 5.0) - Windows Internet Explorer provided by Morrison & Foerster LLP

http://download.oracle.com/javase/1.5.0/docs/api/

java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument

java.beans
Interfaces
AppletInitializer
BeanInfo
Customizer
DesignMode
ExceptionListener
PropertyChangeListener
PropertyEditor
VetoableChangeListener
Visibility
Classes
BeanDescriptor
Beans
DefaultPersistenceDelegate
Encoder
EventHandler
EventSetDescriptor
Expression
FeatureDescriptor
IndexedPropertyChangeEvent

Overview **Package** Class Use Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Java™ 2 Platform Standard Ed. 5.0

Package java.beans

Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.

See:
Description

Interface Summary	
AppletInitializer	This interface is designed to work in collusion with java.beans.Beans.instantiate.
BeanInfo	A bean implementor who wishes to provide explicit information about their bean may provide a BeanInfo class that implements this BeanInfo interface and provides explicit information about the methods, properties, events, etc, of their bean.
Customizer	A customizer class provides a complete custom GUI for customizing a target Java Bean.
DesignMode	This interface is intended to be implemented by, or delegated from, instances of java.beans.beancontext.BeanContext, in order to propagate to its nested hierarchy of java.beans.beancontext.BeanContextChild instances, the current "designTime" property.
ExceptionListener	An ExceptionListener is notified of internal exceptions.
PropertyChangeListener	A "PropertyChange" event gets fired whenever a bean changes a "bound" property.
PropertyEditor	A PropertyEditor class provides support for GUIs that want to allow users to edit a property value of a given type.
VetoableChangeListener	A VetoableChange event gets fired whenever a bean changes a "constrained" property.
Visibility	Under some circumstances a bean may be run on servers where a GUI is not available.

Done

java.beans | Android Developers - Windows Internet Explorer provided by Morrison & Foerster LLP

http://developer.android.com/reference/java/beans/package-summary.html

android developers

English Android.com

search developer docs Search

Home SDK Dev Guide **Reference** Resources Videos Blog

Filter by API Level: 8

android.view.animation
android.view.inputmethod
android.webkit
android.widget
dalvik.bytecode
dalvik.system
java.awt.font
java.beans
java.io
java.lang
java.lang.annotation
java.lang.ref
java.lang.reflect

Interfaces
PropertyChangeListener
Classes
IndexedPropertyChangeEvent
PropertyChangeEvent
PropertyChangeListenerProxy
PropertyChangeSupport

Use Tree Navigation

package
java.beans

Since: API Level 3

Interfaces

PropertyChangeListener	A PropertyChangeListener can subscribe with a event source.

Classes

IndexedPropertyChangeEvent	A type of `PropertyChangeEvent` that indicates that an indexed property has changed.
PropertyChangeEvent	An event that indicates that a constraint or a boundary of a property has changed.
PropertyChangeListenerProxy	The implementation of this listener proxy just delegates the received events to its listener.
PropertyChangeSupport	This utility class

Error on page.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
Interface Summary		**Interfaces**	
Appendable	An object to which `char` sequences and values can be appended.	Appendable	Declares methods to append characters or character sequences.
CharSequence	A `CharSequence` is a readable sequence of `char` values.	CharSequence	This interface represents an ordered set of characters and defines the methods to probe them.
Cloneable	A class implements the `Cloneable` interface to indicate to the `Object.clone()` method that it is legal for that method to make a field-for-field copy of instances of that class.	Cloneable	This (empty) interface must be implemented by all classes that wish to support cloning.
Comparable<T>	This interface imposes a total ordering on the objects of each class that implements it.	Comparable<T>	This interface should be implemented by all classes that wish to define a *natural order* of their instances.
Iterable<T>	Implementing this interface allows an object to be the target of the "foreach" statement.	terable<T>	nstances of classes that implement this interface can be used with the enhanced for loop.
Readable	A `Readable` is a source of characters.	Readable	Represents a sequence of characters that can be ncrementally read (copied) into a `CharBuffer`.
Runnable	The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread.	Runnable	Represents a command that can be executed.
Thread.Uncaught ExceptionHandler	Interface for handlers invoked when a `Thread` abruptly terminates due to an uncaught exception.	Thread.Uncaught ExceptionHandler	mplemented by objects that want to handle cases where a thread is being terminated by an uncaught exception.
Class Summary		**Classes**	
Boolean	The Boolean class wraps a value of the primitive type `boolean` in an object.	Boolean	The wrapper for the primitive type `boolean`.

Exhibit B

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
Byte	The Byte class wraps a value of primitive type byte in an object.	Byte	The wrapper for the primitive type byte.
Character	The Character class wraps a value of the primitive type char in an object.	Character	The wrapper for the primitive type char.
Character.Subset	Instances of this class represent particular subsets of the Unicode character set.	Character.Subset	
Character.Unicode Block	A family of character subsets representing the character blocks in the Unicode specification.	Character.Unicode Block	Represents a block of Unicode characters, as defined by the Unicode 4.0.1 specification.
Class<T>	Instances of the class Class represent classes and interfaces in a running Java application.	Class<T>	The in-memory representation of a Java class.
ClassLoader	A class loader is an object that is responsible forr loading classes.	ClassLoader	Loads classes and resources from a repository.
Compiler	The Compiler class is provided to support Java-to-native-code compilers and related services.	Compiler	Placeholder class for environments which explicitly manage the action of a *Just In Time (JIT)* compiler.
Double	The Double class wraps a value of the primitive type double in an object.	Double	The wrapper for the primitive type double.
Enum<E extends Enum<E>>	This is the common base class of all Java language enumeration types.	Enum<E extends Enum<E>>	The superclass of all enumerated types.
Float	The Float class wraps a value of primitive type float in an object.	Float	The wrapper for the primitive type float.
InheritableThread Local<T>	This class extends ThreadLocal to provide inheritance of values from parent thread to child thread: when a child thread is created, the child receives initial values for all inheritable thread-local variables for which the parent has values.	InheritableThread Local<T>	A thread-local variable whose value is passed from parent to child thread.
Integer	The Integer class wraps a value of the primitive type int in an object.	nteger	The wrapper for the primitive type int.

Exhibit B

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
Long	The `Long` class wraps a value of the primitive type `long` in an object.	Long	The wrapper for the primitive type `long`.
Math	The class `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.	Math	Class Math provides basic math constants and operations such as trigonometric functions, hyperbolic functions, exponential, logarithms, etc.
Number	The abstract class `Number` is the superclass of classes `BigDecimal`, `BigInteger`, `Byte`, `Double`, `Float`, `Integer`, `Long`, and `Short`.	Number	The abstract superclass of the classes which represent numeric base types (that is `Byte`, `Short`, `Integer`, `Long`, `Float`, and `Double`.
Object	Class `Object` is the root of the class hierarchy.	Object	The root class of the Java class hierarchy.
Package	`Package` objects contain version information about the implementation and specification of a Java package.	Package	Contains information about a Java package.
Process	The `ProcessBuilder.start()` and `Runtime.exec` methods create a native process and return an instance of a subclass of `Process` that can be used to control the process and obtain information about it.	Process	Represents an external process.
ProcessBuilder	This class is used to create operating system processes.	ProcessBuilder	Creates operating system processes.
Runtime	Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running.	Runtime	Allows Java applications to interface with the environment in which they are running.
RuntimePermission	This class is for runtime permissions.	RuntimePermission	Represents the permission to execute a runtime-related function.
SecurityManager	The security manager is a class that allows applications to implement a security policy.	SecurityManager	**Warning:** security managers do **not** provide a secure environment for executing untrusted code.
Short	The `Short` class wraps a value of primitive type	Short	The wrapper for the primitive type `short`.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	`short` in an object.		
StackTraceElement	An element in a stack trace, as returned by `Throwable.getStackTrace()`.	StackTraceElement	A representation of a single stack frame.
StrictMath	The class `StrictMath` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.	StrictMath	Class StrictMath provides basic math constants and operations such as trigonometric functions, hyperbolic functions, exponential, logarithms, etc.
String	The `String` class represents character strings.	String	An immutable sequence of characters/code units (`chars`).
StringBuffer	A thread-safe, mutable sequence of characters.	StringBuffer	A modifiable `sequence of characters` for use in creating strings, where all accesses are synchronized.
StringBuilder	A mutable sequence of characters.	StringBuilder	A modifiable `sequence of characters` for use in creating strings.
System	The `System` class contains several useful class fields and methods.	System	Provides access to system-related information and resources including standard input and output.
Thread	A *thread* is a thread of execution in a program.	Thread	A `Thread` is a concurrent unit of execution.
ThreadGroup	A thread group represents a set of threads.	ThreadGroup	`ThreadGroup` is a means of organizing threads into a hierarchical structure.
ThreadLocal<T>	This class provides thread-local variables.	ThreadLocal<T>	mplements a thread-local storage, that is, a variable for which each thread has its own value.
Throwable	The `Throwable` class is the superclass of all errors and exceptions in the Java language.	Throwable	The superclass of all classes which can be thrown by the virtual machine.
Void	The Void class is an uninstantiable placeholder class to hold a reference to the Class object representing the Java keyword void.	Void	Placeholder class for the Java keyword `void`.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
Enum Summary		**Enums**	
Thread.State	A thread state.	Thread.State	A representation of a thread's state.
Exception Summary		**Exceptions**	
Arithmetic Exception	Thrown when an exceptional arithmetic condition has occurred.	ArithmeticException	Thrown when the an invalid arithmetic operation is attempted.
ArrayIndexOut OfBounds Exception	Thrown to indicate that an array has been accessed with an illegal index.	ArrayIndexOutOf BoundsException	Thrown when the an array is indexed with a value less than zero, or greater than or equal to the size of the array.
ArrayStore Exception	Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.	ArrayStore Exception	Thrown when a program attempts to store an element of an incompatible type in an array.
ClassCast Exception	Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.	ClassCastException	Thrown when a program attempts to cast a an object to a type with which it is not compatible.
ClassNotFound Exception	Thrown when an application tries to load in a class through its string name using: The `forName` method in class `Class`.	ClassNotFound Exception	Thrown when a class loader is unable to find a class.
CloneNot Supported Exception	Thrown to indicate that the `clone` method in class `Object` has been called to clone an object, but that the object's class does not implement the `Cloneable` interface.	CloneNotSupported Exception	Thrown when a program attempts to clone an object which does not support the `Cloneable` interface.
EnumConstant NotPresent Exception	Thrown when an application tries to access an enum constant by name and the enum type contains no constant with the specified name.	EnumConstantNot PresentException	Thrown if an `enum` constant does not exist for a particular name.
Exception	The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to	Exception	`Exception` is the superclass of all classes that represent recoverable exceptions.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	catch.		
IllegalAccess Exception	An IllegalAccessException is thrown when an application tries to reflectively create an instance (other than an array), set or get a field, or invoke a method, but the currently executing method does not have access to the definition of the specified class, field, method or constructor.	llegalAccess Exception	Thrown when a program attempts to access a field or method which is not accessible from the location where the reference is made.
IllegalArgumentEx ception	Thrown to indicate that a method has been passed an illegal or inappropriate argument.	llegalArgument Exception	Thrown when a method is invoked with an argument which it can not reasonably deal with.
IllegalMonitor StateException	Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.	llegalMonitorState Exception	Thrown when a monitor operation is attempted when the monitor is not in the correct state, for example when a thread attempts to exit a monitor which it does not own.
IllegalState Exception	Signals that a method has been invoked at an illegal or inappropriate time.	llegalState Exception	Thrown when an action is attempted at a time when the virtual machine is not in the correct state.
IllegalThread StateException	Thrown to indicate that a thread is not in an appropriate state for the requested operation.	llegalThreadState Exception	Thrown when an operation is attempted which is not possible given the state that the executing thread is in.
IndexOutOf BoundsException	Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.	ndexOutOfBounds Exception	Thrown when a program attempts to access a value in an indexable collection using a value which is outside of the range of valid indices.
Instantiation Exception	Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.	nstantiation Exception	Thrown when a program attempts to access a constructor which is not accessible from the location where the reference is made.
Interrupted Exception	Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another	nterrupted Exception	Thrown when a waiting thread is activated before the condition it was waiting for has been satisfied.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	thread interrupts it using the `interrupt` method in class `Thread`.		
NegativeArraySizeException	Thrown if an application tries to create an array with negative size.	NegativeArraySizeException	Thrown when an attempt is made to create an array with a size of less than zero.
NoSuchFieldException	Signals that the class doesn't have a field of a specified name.	NoSuchFieldException	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a field that does not exist.
NoSuchMethodException	Thrown when a particular method cannot be found.	NoSuchMethodException	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a method that does not exist.
NullPointerException	Thrown when an application attempts to use `null` in a case where an object is required.	NullPointerException	Thrown when a program tries to access a field or method of an object or an element of an array when there is no instance or array to use, that is if the object or array points to `null`.
NumberFormatException	Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.	NumberFormatException	Thrown when an invalid value is passed to a string-to-number conversion method.
RuntimeException	`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.	RuntimeException	`RuntimeException` is the superclass of all classes that represent exceptional conditions which occur as a result of executing an application in the virtual machine.
SecurityException	Thrown by the security manager to indicate a security violation.	SecurityException	Thrown when a security manager check fails.
StringIndexOutOfBoundsException	Thrown by `String` methods to indicate that an index is either negative or greater than the size of the string.	StringIndexOutOfBoundsException	Thrown when the a string is indexed with a value less than zero, or greater than or equal to the size of the array.
TypeNotPresentException	Thrown when an application tries to access a type using a string representing the type's	TypeNotPresentException	Thrown when a program tries to access a class, nterface, enum or annotation type through a string that contains the type's name and the type cannot be

Exhibit B

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
	name, but no definition for the type with the specified name can be found.		found.
Unsupported Operation Exception	Thrown to indicate that the requested operation is not supported.	Unsupported OperationException	Thrown when an unsupported operation is attempted.

Error Summary

AbstractMethod Error	Thrown when an application tries to call an abstract method.
AssertionError	Thrown to indicate that an assertion has failed.
ClassCircularity Error	Thrown when a circularity has been detected while initializing a class.
ClassFormat Error	Thrown when the Java Virtual Machine attempts to read a class file and determines that the file is malformed or otherwise cannot be interpreted as a class file.
Error	An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch.
ExceptionIn InitializerError	Signals that an unexpected exception has occurred in a static initializer.

Errors

AbstractMethod Error	Thrown by the virtual machine when an abstract method is called.
AssertionError	Thrown when an assertion has failed.
ClassCircularity Error	Thrown when the virtual machine notices that an attempt is made to load a class which would directly or ndirectly inherit from one of its subclasses.
ClassFormatError	Thrown by a class loader when a class file has an llegal format or if the data that it contains can not be nterpreted as a class.
Error	`Error` is the superclass of all classes that represent unrecoverable errors.
ExceptionInInitialize rError	Thrown when an exception occurs during class nitialization.

Exhibit B

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
IllegalAccessError	Thrown if an application attempts to access or modify a field, or to call a method that it does not have access to.	llegalAccessError	Thrown when the virtual machine notices that a program tries access a field which is not accessible from where it is referenced.
IncompatibleClassChangeError	Thrown when an incompatible class change has occurred to some class definition.	ncompatibleClassChangeError	`IncompatibleClassChangeError` is the superclass of all classes which represent errors that occur when nconsistent class files are loaded into the same running image.
InstantiationError	Thrown when an application tries to use the Java `new` construct to instantiate an abstract class or an interface.	nstantiationError	Thrown when the virtual machine notices that a program tries to create a new instance of a class which has no visible constructors from the location where `new` s invoked.
InternalError	Thrown to indicate some unexpected internal error has occurred in the Java Virtual Machine.	nternalError	Thrown when the virtual machine notices that it has gotten into an undefined state.
LinkageError	Subclasses of `LinkageError` indicate that a class has some dependency on another class; however, the latter class has incompatibly changed after the compilation of the former class.	LinkageError	`LinkageError` is the superclass of all error classes that occur when loading and linking class files.
NoClassDefFoundError	Thrown if the Java Virtual Machine or a `ClassLoader` instance tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the `new` expression) and no definition of the class could be found.	NoClassDefFoundError	Thrown when the virtual machine is unable to locate a class which it has been asked to load.
NoSuchFieldError	Thrown if an application tries to access or modify a specified field of an object, and that object no longer has that field.	NoSuchFieldError	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a field that does not exist.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)		Android APIs (java.lang)	
NoSuchMethod Error	Thrown if an application tries to call a specified method of a class (either static or instance), and that class no longer has a definition of that method.	NoSuchMethod Error	Thrown when the virtual machine notices that a program tries to reference, on a class or object, a method that does not exist.
OutOfMemory Error	Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.	OutOfMemoryError	Thrown when a request for memory is made that can not be satisfied using the available platform resources.
StackOverflow Error	Thrown when a stack overflow occurs because an application recurses too deeply.	StackOverflowError	Thrown when the depth of the callstack of the running program excedes some platform or virtual machine specific limit.
ThreadDeath	An instance of `ThreadDeath` is thrown in the victim thread when the `stop` method with zero arguments in class `Thread` is called.	ThreadDeath	ThreadDeath is thrown when a thread stops executing.
UnknownError	Thrown when an unknown but serious exception has occurred in the Java Virtual Machine.	UnknownError	Thrown when the virtual machine must throw an error which does not match any known exceptional condition.
UnsatisfiedLink Error	Thrown if the Java Virtual Machine cannot find an appropriate native-language definition of a method declared `native`.	UnsatisfiedLinkError	Thrown when an attempt is made to invoke a native for which an implementation could not be found.
Unsupported ClassVersion Error	Thrown when the Java Virtual Machine attempts to read a class file and determines that the major and minor version numbers in the file are not supported.	UnsupportedClass VersionError	Thrown when an attempt is made to load a class with a format version that is not supported by the virtual machine.
VerifyError	Thrown when the "verifier" detects that a class file, though well formed, contains some sort of internal inconsistency or security problem.	VerifyError	Thrown when the virtual machine notices that an attempt is made to load a class which does not pass the class verification phase.
VirtualMachine Error	Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.	VirtualMachineError	`VirtualMachineError` is the superclass of all error classes that occur during the operation of the virtual machine.

Exhibit B

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang)	Android APIs (java.lang)
Annotation Types Summary **Deprecated** — A program element annotated @Deprecated is one that programmers are discouraged from using, typically because it is dangerous, or because a better alternative exists. **Override** — Indicates that a method declaration is intended to override a method declaration in a superclass. **Suppress Warnings** — Indicates that the named compiler warnings should be suppressed in the annotated element (and in all program elements contained in the annotated element).	

Exhibit C

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.io)	Android APIs (java.io)
Interface Summary	**Interfaces**
Closeable — A `Closeable` is a source or destination of data that can be closed.	Closeable — Defines an interface for classes that can (or need to) be closed once they are not used any longer.
DataInput — The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.	DataInput — Defines an interface for classes that are able to read typed data from some source.
DataOutput — The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.	DataOutput — Defines an interface for classes that are able to write typed data to some target.
Externalizable — Only the identity of the class of an Externalizable instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances.	Externalizable — Defines an interface for classes that want to be serializable, but have their own binary representation.
FileFilter — A filter for abstract pathnames.	FileFilter — An interface for filtering `File` objects based on their names or other nformation.
FilenameFilter — Instances of classes that implement this interface are used to filter filenames.	FilenameFilter — An interface for filtering `File` objects based on their names or the directory they reside in.
Flushable — A `Flushable` is a destination of data that can be flushed.	Flushable — Defines an interface for classes that can (or need to) be flushed, typically before some output processing is considered to be finished and the object gets closed.

Exhibit C

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
ObjectInput	ObjectInput extends the DataInput interface to include the reading of objects.	ObjectInput	Defines an interface for classes that allow reading serialized objects.
ObjectInputValidation	Callback interface to allow validation of objects within a graph.	ObjectInputValidation	A callback interface for post-deserialization checks on objects.
ObjectOutput	ObjectOutput extends the DataOutput interface to include writing of objects.	ObjectOutput	Defines an interface for classes that allow reading serialized objects.
ObjectStreamConstants	Constants written into the Object Serialization Stream.	ObjectStreamConstants	A helper interface with constants used by the serialization implementation.
Serializable	Serializability of a class is enabled by the class implementing the java.io.Serializable interface.	Serializable	An empty marker interface for classes that want to support serialization and deserialization based on the `ObjectOutputStream` and `ObjectInputStream` classes.
Class Summary		**Classes**	
BufferedInputStream	A `BufferedInputStream` adds functionality to another input stream-namely, the ability to buffer the input and to support the `mark` and `reset` methods.	BufferedInputStream	Wraps an existing `InputStream` and *buffers* the input.
BufferedOutputStream	The class implements a buffered output stream.	BufferedOutputStream	Wraps an existing `OutputStream` and *buffers* the output.
BufferedReader	Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.	BufferedReader	Wraps an existing `Reader` and *buffers* the input.
BufferedWriter	Write text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters,	BufferedWriter	Wraps an existing `Writer` and *buffers* the output.

Exhibit C

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
	arrays, and strings.		
ByteArrayInputStream	A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream.	ByteArrayInputStream	A specialized `InputStream` for reading the contents of a byte array.
ByteArrayOutputStream	This class implements an output stream in which the data is written into a byte array.	ByteArrayOutputStream	A specialized `OutputStream` for class for writing content to an (internal) byte array.
CharArrayReader	This class implements a character buffer that can be used as a character-input stream.	CharArrayReader	A specialized `Reader` for reading the contents of a char array.
CharArrayWriter	This class implements a character buffer that can be used as an Writer.	CharArrayWriter	A specialized `Writer` for class for writing content to an (internal) char array.
		Console	Provides access to the console, if available.
DataInputStream	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.	DataInputStream	Wraps an existing `InputStream` and reads typed data from it.
DataOutputStream	A data output stream lets an application write primitive Java data types to an output stream in a portable way.	DataOutputStream	Wraps an existing `OutputStream` and writes typed data to it.
File	An abstract representation of file and directory pathnames.	File	An "abstract" representation of a file system entity identified by a pathname.
FileDescriptor	Instances of the file descriptor class serve as an opaque handle to the underlying machine-specific structure representing an open file, an open socket, or another source or sink of bytes.	FileDescriptor	The lowest-level representation of a file, device, or socket.

Exhibit C

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
FileInputStream	A `FileInputStream` obtains input bytes from a file in a file system.	FileInputStream	A specialized `InputStream` that reads from a file in the file system.
FileOutputStream	A file output stream is an output stream for writing data to a `File` or to a `FileDescriptor`.	FileOutputStream	A specialized `OutputStream` that writes to a file in the file system.
FilePermission	This class represents access to a file or directory.	FilePermission	A permission for accessing a file or directory.
FileReader	Convenience class for reading character files.	FileReader	A specialized `Reader` that reads from a file in the file system.
FileWriter	Convenience class for writing character files.	FileWriter	A specialized `Writer` that writes to a file in the file system.
FilterInputStream	A `FilterInputStream` contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.	FilterInputStream	Wraps an existing `InputStream` and performs some transformation on the input data while it is being read.
FilterOutputStream	This class is the superclass of all classes that filter output streams.	FilterOutputStream	Wraps an existing `OutputStream` and performs some transformation on the output data while it is being written.
FilterReader	Abstract class for reading filtered character streams.	FilterReader	Wraps an existing `Reader` and performs some transformation on the input data while it is being read.

Exhibit C

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
FilterWriter	Abstract class for writing filtered character streams.	FilterWriter	Wraps an existing `Writer` and performs some transformation on the output data while it is being written.
InputStream	This abstract class is the superclass of all classes representing an input stream of bytes.	nputStream	The base class for all input streams.
InputStreamReader	An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified `charset`.	nputStreamReader	A class for turning a byte stream into a character stream.
LineNumberInputStream	**Deprecated.** *This class incorrectly assumes that bytes adequately represent characters.*	LineNumberInputStream	*This class is deprecated. Use* `LineNumberReader`
LineNumberReader	A buffered character-input stream that keeps track of line numbers.	LineNumberReader	Wraps an existing `Reader` and counts the line terminators encountered while reading the data.
ObjectInputStream	An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.	ObjectInputStream	A specialized `InputStream` that is able to read (deserialize) Java objects as well as primitive data types (int, byte, char etc.).
ObjectInputStream.GetField	Provide access to the persistent fields read from the input stream.	ObjectInputStream.GetField	GetField is an inner class that provides access to the persistent fields read from the source stream.
ObjectOutputStream	An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.	ObjectOutputStream	A specialized `OutputStream` that is able to write (serialize) Java objects as well as primitive data types (int, byte, char etc.).

Exhibit C

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
ObjectOutputStream.PutField	Provide programmatic access to the persistent fields to be written to ObjectOutput.	ObjectOutputStream.PutField	PutField is an inner class to provide access to the persistent fields that are written to the target stream.
ObjectStreamClass	Serialization's descriptor for classes.	ObjectStreamClass	Represents a descriptor for identifying a class during serialization and deserialization.
ObjectStreamField	A description of a Serializable field from a Serializable class.	ObjectStreamField	Describes a field for the purpose of serialization.
OutputStream	This abstract class is the superclass of all classes representing an output stream of bytes.	OutputStream	The base class for all output streams.
OutputStreamWriter	An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are encoded into bytes using a specified `charset`.	OutputStreamWriter	A class for turning a character stream into a byte stream.
PipedInputStream	A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream.	PipedInputStream	Receives information from a communications pipe.
PipedOutputStream	A piped output stream can be connected to a piped input stream to create a communications pipe.	PipedOutputStream	Places information on a communications pipe.
PipedReader	Piped character-input streams.	PipedReader	Receives information on a communications pipe.
PipedWriter	Piped character-output streams.	PipedWriter	Places information on a communications pipe.

Exhibit C

Java™ 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
PrintStream	A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently.	PrintStream	Wraps an existing `OutputStream` and provides convenience methods for writing common data types in a human readable format.
PrintWriter	Print formatted representations of objects to a text-output stream.	PrintWriter	Wraps either an existing `OutputStream` or an existing `Writer` and provides convenience methods for printing common data types in a human readable format.
PushbackInputStream	A `PushbackInputStream` adds functionality to another input stream, namely the ability to "push back" or "unread" one byte.	PushbackInputStream	Wraps an existing `InputStream` and adds functionality to "push back" bytes that have been read, so that they can be read again.
PushbackReader	A character-stream reader that allows characters to be pushed back into the stream.	PushbackReader	Wraps an existing `Reader` and adds functionality to "push back" characters that have been read, so that they can be read again.
RandomAccessFile	Instances of this class support both reading and writing to a random access file.	RandomAccessFile	Allows reading from and writing to a file in a random-access manner.

Exhibit C

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
Reader	Abstract class for reading character streams.	Reader	The base class for all readers.
SequenceInputStream	A `SequenceInputStream` represents the logical concatenation of other input streams.	SequenceInputStream	Concatenates two or more existing `InputStream`s.
SerializablePermission	This class is for Serializable permissions.	SerializablePermission	Is used to enable access to potentially unsafe serialization operations.
StreamTokenizer	The `StreamTokenizer` class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time.	StreamTokenizer	Parses a stream into a set of defined tokens, one at a time.
StringBufferInputStream	**Deprecated.** *This class does not properly convert characters into bytes.*	StringBufferInputStream	*This class is deprecated. Use `StringReader`*
StringReader	A character stream whose source is a string.	StringReader	A specialized `Reader` that reads characters from a `String` in a sequential manner.
StringWriter	A character stream that collects its output in a string buffer, which can then be used to construct a string.	StringWriter	A specialized `Writer` that writes characters to a `StringBuffer` in a sequential manner, appending them in the process.
Writer	Abstract class for writing to character streams.	Writer	The base class for all writers.

Exhibit C

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
Exception Summary		**Exceptions**	
CharConversionException	Base class for character conversion exceptions.	CharConversionException	The top level class for character conversion exceptions.
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input.	EOFException	Thrown when a program encounters the end of a file or stream during an nput operation.
FileNotFoundException	Signals that an attempt to open the file denoted by a specified pathname has failed.	FileNotFoundException	Thrown when a file specified by a program cannot be found.
InterruptedIOException	Signals that an I/O operation has been interrupted.	nterruptedIOException	Signals that a blocking I/O operation has been interrupted.
InvalidClassException	Thrown when the Serialization runtime detects one of the following problems with a Class.	nvalidClassException	Signals a problem during the serialization or or deserialization of an object.
InvalidObjectException	Indicates that one or more deserialized objects failed validation tests.	nvalidObjectException	Signals that, during deserialization, the validation of an object has failed.
IOException	Signals that an I/O exception of some sort has occurred.	OException	Signals a general, I/O-related error.
NotActiveException	Thrown when serialization or deserialization is not active.	NotActiveException	Signals that a serialization-related method has been invoked in the wrong place.
NotSerializableException	Thrown when an instance is required to have a Serializable interface.	NotSerializableException	Signals that an object that is not serializable has been passed into the `ObjectOutput.writeObject()` method.

Exhibit C

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.io)		Android APIs (java.io)	
ObjectStreamException	Superclass of all exceptions specific to Object Stream classes.	ObjectStreamException	Signals some sort of problem during either serialization or deserialization of objects.
OptionalDataException	Exception indicating the failure of an object read operation due to unread primitive data, or the end of data belonging to a serialized object in the stream.	OptionalDataException	Signals that the `ObjectInputStream` class encountered a primitive type (`int`, `char` etc.) instead of an object nstance in the input stream.
StreamCorruptedException	Thrown when control information that was read from an object stream violates internal consistency checks.	StreamCorruptedException	Signals that the `readObject()` method could not read an object due to missing information (for example, a cyclic reference that doesn't match a previous instance, or a missing class descriptor for the object to be loaded).
SyncFailedException	Signals that a sync operation has failed.	SyncFailedException	Signals that the `sync()` method has failed to complete.
UnsupportedEncodingException	The Character Encoding is not supported.	UnsupportedEncodingException	Thrown when a program asks for a particular character converter that is unavailable.
UTFDataFormatException	Signals that a malformed string in modified UTF-8 format has been read in a data input stream or by any class that implements the data input interface.	UTFDataFormatException	Signals that an incorrectly encoded UTF-8 string has been encountered, most likely while reading some `DataInputStream`.
WriteAbortedException	Signals that one of the ObjectStreamExceptions was thrown during a write operation.	WriteAbortedException	Signals that the `readObject()` method has detected an exception marker in the input stream.

Exhibit D

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Interface Summary		**Interfaces**	
Certificate	**Deprecated.** *A new certificate handling package is created in the Java 2 platform.*	Certificate	*This interface is deprecated. Replaced by behavior in `java.security.cert`*
DomainCombiner	A `DomainCombiner` provides a means to dynamically update the ProtectionDomains associated with the current `AccessControlContext`.	DomainCombiner	`DomainCombiner` is used to update and optimize `ProtectionDomain`s from an `AccessControlContext`.
Guard	This interface represents a guard, which is an object that is used to protect access to another object.	Guard	`Guard` implementors protect access to other objects.
Key	The Key interface is the top-level interface for all keys.	Key	`Key` is the common interface for all keys.
KeyStore.Entry	A marker interface for `KeyStore` entry types.	KeyStore.Entry	`Entry` is the common marker interface for a `KeyStore` entry.
KeyStore.LoadStoreParameter	A marker interface for `KeyStore` `load` and `store` parameters.	KeyStore.LoadStoreParameter	`LoadStoreParameter` represents a parameter that specifies how a `KeyStore` can be loaded and stored.
KeyStore.ProtectionParameter	A marker interface for keystore protection parameters.	KeyStore.ProtectionParameter	`ProtectionParameter` is a marker interface for protection parameters.
		Policy.Parameters	A marker interface for Policy parameters.
Principal	This interface represents the abstract notion of a principal, which can be used to represent any entity, such as an individual, a corporation, and a login id.	Principal	`Principal`s are objects which have identities.

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
PrivateKey	A private key.	PrivateKey	`PrivateKey` is the common interface for private keys.
PrivilegedAction<T>	A computation to be performed with privileges enabled.	PrivilegedAction<T>	`PrivilegedAction` represents an action that can be executed privileged regarding access control.
PrivilegedExceptionAction<T>	A computation to be performed with privileges enabled, that throws one or more checked exceptions.	PrivilegedExceptionAction<T>	`PrivilegedAction` represents an action, that can be executed privileged regarding access control.
PublicKey	A public key.	PublicKey	`PublicKey` is the common interface for public keys.
Class Summary		**Classes**	
AccessControlContext	An AccessControlContext is used to make system resource access decisions based on the context it encapsulates.	AccessControlContext	`AccessControlContext` encapsulates the `ProtectionDomain`s on which access control decisions are based.
AccessController	The AccessController class is used for access control operations and decisions.	AccessController	`AccessController` provides static methods to perform access control checks and privileged operations.
AlgorithmParameterGenerator	The `AlgorithmParameterGenerator` class is used to generate a set of parameters to be used with a certain algorithm.	AlgorithmParameterGenerator	`AlgorithmParameterGenerator` is an engine class which is capable of generating parameters for the algorithm it was initialized with.
AlgorithmParameterGeneratorSpi	This class defines the *Service Provider Interface* (**SPI**) for the	AlgorithmParameterGeneratorSpi	`AlgorithmParameterGeneratorSpi` is the Service Provider Interface (SPI) definition for

Exhibit D

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
	`AlgorithmParameterGenerator` class, which is used to generate a set of parameters to be used with a certain algorithm.		`AlgorithmParameterGenerator`.
AlgorithmParameters	This class is used as an opaque representation of cryptographic parameters.	AlgorithmParameters	`AlgorithmParameters` is an engine class which provides algorithm parameters.
AlgorithmParametersSpi	This class defines the *Service Provider Interface* (**SPI**) for the `AlgorithmParameters` class, which is used to manage algorithm parameters.	AlgorithmParametersSpi	`AlgorithmParametersSpi` is the Service Provider Interface (SPI) definition for `AlgorithmParameters`.
AllPermission	The AllPermission is a permission that implies all other permissions.	AllPermission	`AllPermission` represents the permission to perform any operation.
AuthProvider	This class defines login and logout methods for a provider.	AuthProvider	`AuthProvider` is an abstract superclass for Java Security `Provider` which provide login and logout.
BasicPermission	The BasicPermission class extends the Permission class, and can be used as the base class for permissions that want to follow the same naming convention as BasicPermission.	BasicPermission	`BasicPermission` is the common base class of all permissions which have a name but no action lists.
CodeSigner	This class encapsulates information about a code signer.	CodeSigner	`CodeSigner` represents a signer of code.
CodeSource	This class extends the concept of a codebase to encapsulate not only the location (URL) but also the certificate chains that were used to verify signed code originating from that location.	CodeSource	`CodeSource` encapsulates the location from where code is loaded and the certificates that were used to verify that code.

Exhibit D

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
DigestInputStream	A transparent stream that updates the associated message digest using the bits going through the stream.	DigestInputStream	`DigestInputStream` is a `FilterInputStream` which maintains an associated message digest.
DigestOutputStream	A transparent stream that updates the associated message digest using the bits going through the stream.	DigestOutputStream	`DigestOutputStream` is a `FilterOutputStream` which maintains an associated message digest.
GuardedObject	A GuardedObject is an object that is used to protect access to another object.	GuardedObject	`GuardedObject` controls access to an object, by checking all requests for the object with a `Guard`.
Identity	**Deprecated.** *This class is no longer used.*	Identity	*This class is deprecated. The functionality of this class has been replace by `Principal`, `KeyStore` and the `java.security.cert` package.*
IdentityScope	**Deprecated.** *This class is no longer used.*	IdentityScope	*This class is deprecated. The functionality of this class has been replace by `Principal`, `KeyStore` and the `java.security.cert` package.*

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
KeyFactory	Key factories are used to convert *keys* (opaque cryptographic keys of type `Key`) into *key specifications* (transparent representations of the underlying key material), and vice versa.	KeyFactory	`KeyFactory` is an engine class that can be used to translate between public and private key objects and convert keys between their external representation, that can be easily transported and their internal representation.
KeyFactorySpi	This class defines the *Service Provider Interface* (**SPI**) for the `KeyFactory` class.	KeyFactorySpi	`KeyFactorySpi` is the Service Provider Interface (SPI) definition for `KeyFactory`.
KeyPair	This class is a simple holder for a key pair (a public key and a private key).	KeyPair	`KeyPair` is a container for a public key and a private key.
KeyPairGenerator	The KeyPairGenerator class is used to generate pairs of public and private keys.	KeyPairGenerator	`KeyPairGenerator` is an engine class which is capable of generating a private key and its related public key utilizing the algorithm it was initialized with.
KeyPairGeneratorSpi	This class defines the *Service Provider Interface* (**SPI**) for the `KeyPairGenerator` class, which is used to generate pairs of public and private keys.	KeyPairGeneratorSpi	`KeyPairGeneratorSpi` is the Service Provider Interface (SPI) definition for `KeyPairGenerator`.
KeyRep	Standardized representation for serialized Key objects.	KeyRep	`KeyRep` is a standardized representation for serialized `Key` objects.
KeyStore	This class represents a storage facility for cryptographic keys and certificates.	KeyStore	`KeyStore` is responsible for maintaining cryptographic keys and their owners.

Exhibit D

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
KeyStore.Builder	A description of a to-be-instantiated KeyStore object.	KeyStore.Builder	`Builder` is used to construct new instances of `KeyStore`.
KeyStore.Callback HandlerProtection	A ProtectionParameter encapsulating a CallbackHandler.	KeyStore.Callback HandlerProtection	`CallbackHandlerProtection` is a `ProtectionParameter` that encapsulates a `CallbackHandler`.
KeyStore.Password Protection	A password-based implementation of `ProtectionParameter`.	KeyStore.Password Protection	`PasswordProtection` is a `ProtectionParameter` that protects a `KeyStore` using a password.
KeyStore.PrivateKey Entry	A `KeyStore` entry that holds a `PrivateKey` and corresponding certificate chain.	KeyStore.PrivateKey Entry	`PrivateKeyEntry` represents a `KeyStore` entry that holds a private key.
KeyStore.SecretKey Entry	A `KeyStore` entry that holds a `SecretKey`.	KeyStore.SecretKey Entry	`SecretKeyEntry` represents a `KeyStore` entry that holds a secret key.
KeyStore.Trusted CertificateEntry	A `KeyStore` entry that holds a trusted `Certificate`.	KeyStore.Trusted CertificateEntry	`TrustedCertificateEntry` represents a `KeyStore` entry that holds a trusted certificate.
KeyStoreSpi	This class defines the *Service Provider Interface* (**SPI**) for the `KeyStore` class.	KeyStoreSpi	`KeyStoreSpi` is the Service Provider Interface (SPI) definition for `KeyStore`.
MessageDigest	This MessageDigest class provides applications the functionality of a message	MessageDigest	Uses a one-way hash function to turn an arbitrary number of bytes into a fixed-

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
	digest algorithm, such as MD5 or SHA.		length byte sequence.
MessageDigestSpi	This class defines the *Service Provider Interface* (**SPI**) for the `MessageDigest` class, which provides the functionality of a message digest algorithm, such as MD5 or SHA.	MessageDigestSpi	`MessageDigestSpi` is the Service Provider Interface (SPI) definition for `MessageDigest`.
Permission	Abstract class for representing access to a system resource.	Permission	`Permission` is the common base class of all permissions that participate in the access control security framework around `AccessController` and `AccessControlContext`.
PermissionCollection	Abstract class representing a collection of Permission objects.	PermissionCollection	`PermissionCollection` is the common base class for all collections that provide a convenient method for determining whether or not a given permission is implied by any of the permissions present in this collection.
Permissions	This class represents a heterogeneous collection of Permissions.	Permissions	`Permissions` represents a `PermissionCollection` where the contained permissions can be of different types.
Policy	This is an abstract class for representing the system security policy for a Java application environment (specifying which permissions are available for code from various sources).	Policy	`Policy` is the common super type of classes which represent a system security policy.
		PolicySpi	Represents the Service Provider Interface (SPI) for java.security.Policy class.
ProtectionDomain	This ProtectionDomain class encapsulates the characteristics of a domain, which encloses a	ProtectionDomain	`ProtectionDomain` represents all permissions that are granted to a specific

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
	set of classes whose instances are granted a set of permissions when being executed on behalf of a given set of Principals.		code source.
Provider	This class represents a "provider" for the Java Security API, where a provider implements some or all parts of Java Security.	Provider	`Provider` is the abstract superclass for all security providers in the Java security infrastructure.
Provider.Service	The description of a security service.	Provider.Service	`Service` represents a service in the Java Security infrastructure.
SecureClassLoader	This class extends ClassLoader with additional support for defining classes with an associated code source and permissions which are retrieved by the system policy by default.	SecureClassLoader	`SecureClassLoader` represents a `ClassLoader` which associates the classes it loads with a code source and provide mechanisms to allow the relevant permissions to be retrieved.
SecureRandom	This class provides a cryptographically strong random number generator (RNG).	SecureRandom	This class generates cryptographically secure pseudo-random numbers.
SecureRandomSpi	This class defines the *Service Provider Interface* (**SPI**) for the `SecureRandom` class.	SecureRandomSpi	`SecureRandomSpi` is the *Service Provider Interface* (**SPI**) definition for `SecureRandom`.
Security	This class centralizes all security properties and common security methods.	Security	`Security` is the central class in the Java Security API.
SecurityPermission	This class is for security permissions.	SecurityPermission	`SecurityPermission` objects guard access to the mechanisms which implement security.

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Signature	This Signature class is used to provide applications the functionality of a digital signature algorithm.	Signature	`Signature` is an engine class which is capable of creating and verifying digital signatures, using different algorithms that have been registered with the `Security` class.
SignatureSpi	This class defines the *Service Provider Interface* (**SPI**) for the `Signature` class, which is used to provide the functionality of a digital signature algorithm.	SignatureSpi	`SignatureSpi` is the *Service Provider Interface* (**SPI**) definition for `Signature`.
SignedObject	SignedObject is a class for the purpose of creating authentic runtime objects whose integrity cannot be compromised without being detected.	SignedObject	A `SignedObject` instance acts as a container for another object.
Signer	**Deprecated.** *This class is no longer used.*	Signer	*This class is deprecated. Replaced by behavior in `java.security.cert` package and `Principal`*
Timestamp	This class encapsulates information about a signed timestamp.	Timestamp	`Timestamp` represents a signed time stamp.
UnresolvedPermission	The UnresolvedPermission class is used to hold Permissions that were "unresolved" when the Policy was initialized.	UnresolvedPermission	An `UnresolvedPermission` represents a `Permission` whose type should be resolved lazy and not during initialization time of the `Policy`.

Enum Summary		**Enums**	
KeyRep.Type	Key type.	KeyRep.Type	`Type` enumerates the supported key types.

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
Exception Summary		**Exceptions**	
AccessControlException	This exception is thrown by the AccessController to indicate that a requested access (to a critical system resource such as the file system or the network) is denied.	AccessControlException	AccessControlException is thrown if the access control infrastructure denies protected access due to missing permissions.
DigestException	This is the generic Message Digest exception.	DigestException	DigestException is a general message digest exception.
GeneralSecurityException	The GeneralSecurityException class is a generic security exception class that provides type safety for all the security-related exception classes that extend from it.	GeneralSecurityException	GeneralSecurityException is a general security exception and the superclass for all security specific exceptions.
InvalidAlgorithmParameterException	This is the exception for invalid or inappropriate algorithm parameters.	InvalidAlgorithmParameterException	InvalidAlgorithmParameterException indicates the occurrence of invalid algorithm parameters.
InvalidKeyException	This is the exception for invalid Keys (invalid encoding, wrong length, uninitialized, etc).	InvalidKeyException	InvalidKeyException indicates exceptional conditions, caused by an invalid key.
InvalidParameterException	This exception, designed for use by the JCA/JCE engine classes, is thrown when an invalid parameter is passed to a method.	InvalidParameterException	InvalidParameterException indicates exceptional conditions, caused by invalid parameters.
KeyException	This is the basic key exception.	KeyException	KeyException is the common superclass of all key related exceptions.

Exhibit D

Java™ 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
KeyManagement Exception	This is the general key management exception for all operations dealing with key management.	KeyManagement Exception	`KeyManagementException` is a general exception, thrown to indicate an exception during processing an operation concerning key management.
KeyStoreException	This is the generic KeyStore exception.	KeyStoreException	`KeyStoreException` is a general `KeyStore` exception.
NoSuchAlgorithm Exception	This exception is thrown when a particular cryptographic algorithm is requested but is not available in the environment.	NoSuchAlgorithm Exception	`NoSuchAlgorithmException` indicates that a requested algorithm could not be found.
NoSuchProviderException	This exception is thrown when a particular security provider is requested but is not available in the environment.	NoSuchProvider Exception	`NoSuchProviderException` indicates that a requested security provider could not be found.
PrivilegedActionException	This exception is thrown by `doPrivileged(PrivilegedExceptionAction)` and `doPrivileged(PrivilegedExceptionAction, AccessControlContext context)` to indicate that the action being performed threw a checked exception.	PrivilegedAction Exception	`PrivilegedActionException` wraps exceptions which are thrown from within privileged operations.
ProviderException	A runtime exception for Provider exceptions (such as misconfiguration errors or unrecoverable internal errors), which may be subclassed by Providers to throw specialized, provider-specific runtime errors.	ProviderException	`ProviderException` is a general exception, thrown by security `Providers`.
SignatureException	This is the generic Signature exception.	SignatureException	`SignatureException` is a general `Signature` exception.

Exhibit D

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.security)		Android APIs (java.security)	
UnrecoverableEntry Exception	This exception is thrown if an entry in the keystore cannot be recovered.	UnrecoverableEntry Exception	`UnrecoverableEntryException` indicates, that a `KeyStore.Entry` cannot be recovered from a `KeyStore`.
UnrecoverableKey Exception	This exception is thrown if a key in the keystore cannot be recovered.	UnrecoverableKey Exception	`UnrecoverableKeyException` indicates, that a key cannot be recovered from a `KeyStore`.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

java.lang
Class Runtime
 	
`java.lang.Object`
  └ **`java.lang.Runtime`**
 	
`public class` **`Runtime`**
`extends Object`
 	
Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the `getRuntime` method.
 	
An application cannot create its own instance of this class.
 	
Since:
JDK1.0
See Also:
`getRuntime()`
	public class
	Runtime
	
	extends Object
	java.lang.Object
	↳java.lang.Runtime
	
	Class Overview
	
	Allows Java applications to interface with the environment in which they are running. Applications can not create an instance of this class, but they can get a singleton instance by invoking `getRuntime()`.
	
	See Also
	`System`

Method Summary
 	
`void` **`addShutdownHook`**`(Thread hook)`
Registers a new virtual-machine shutdown hook.
 	
`int` **`availableProcessors`**`()`
Returns the number of processors available to the
	Summary
	Public Methods
	
	void addShutdownHook(Thread hook)
	Registers a virtual-machine shutdown hook.
	
	int availableProcessors()
	Returns the number of processors available to the virtual machine.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)		Android APIs (java.lang.Runtime)	
	Java virtual machine.		
Process	exec(String command) Executes the specified string command in a separate process.	Process	exec(String[] progArray, String[] envp) Executes the specified command and its arguments in a separate native process.
Process	exec(String[] cmdarray) Executes the specified command and arguments in a separate process.	Process	exec(String prog, String[] envp, File directory) Executes the specified program in a separate native process.
Process	exec(String[] cmdarray, String[] envp) Executes the specified command and arguments in a separate process with the specified environment.	Process	exec(String[] progArray, String[] envp, File directory) Executes the specified command and its arguments in a separate native process.
Process	exec(String[] cmdarray, String[] envp, File dir) Executes the specified command and arguments in a separate process with the specified environment and working directory.	Process	exec(String prog, String[] envp) Executes the specified program in a separate native process.
Process	exec(String command, String[] envp) Executes the specified string command in a separate process with the specified environment.	Process	exec(String prog) Executes the specified program in a separate native process.
Process	exec(String command, String[] envp, File dir) Executes the specified string command in a separate process with the specified environment and working directory.	Process	exec(String[] progArray) Executes the specified command and its arguments in a separate native process.

Exhibit E

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)		Android APIs (java.lang.Runtime)	
`void`	`exit(int status)` Terminates the currently running Java virtual machine by initiating its shutdown sequence.	void	exit(int code) Causes the virtual machine to stop running and the program to exit.
`long`	`freeMemory()` Returns the amount of free memory in the Java Virtual Machine.	long	freeMemory() Returns the amount of free memory resources which are available to the running program.
`void`	`gc()` Runs the garbage collector.	void	gc() Indicates to the virtual machine that it would be a good time to run the garbage collector.
`InputStream`	`getLocalizedInputStream(InputStream in)` **Deprecated.** *As of JDK 1.1, the preferred way to translate a byte stream in the local encoding into a character stream in Unicode is via the `InputStreamReader` and `BufferedReader` classes.*	InputStream	getLocalizedInputStream(InputStream stream) *This method is deprecated. Use `InputStreamReader`.*
`OutputStream`	`getLocalizedOutputStream(OutputStream out)` **Deprecated.** *As of JDK 1.1, the preferred way to translate a Unicode character stream into a byte stream in the local encoding is via the `OutputStreamWriter`, `BufferedWriter`, and `PrintWriter` classes.*	OutputStream	getLocalizedOutputStream(OutputStream stream) *This method is deprecated. Use `OutputStreamWriter`.*
`static Runtime`	`getRuntime()` Returns the runtime object associated with the current Java application.	static Runtime	getRuntime() Returns the single `Runtime` instance.
`void`	`halt(int status)` Forcibly terminates the currently running Java virtual machine.	void	halt(int code) Causes the virtual machine to stop running, and the program to exit.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)		Android APIs (java.lang.Runtime)	
`void`	`load(String filename)` Loads the specified filename as a dynamic library.	void	load(String pathName) Loads and links the dynamic library that is identified through the specified path.
`void`	`loadLibrary(String libname)` Loads the dynamic library with the specified library name.	void	loadLibrary(String libName) Loads and links the library with the specified name.
`long`	`maxMemory()` Returns the maximum amount of memory that the Java virtual machine will attempt to use.	long	maxMemory() Returns the maximum amount of memory that may be used by the virtual machine, or `Long.MAX_VALUE` if there is no such limit.
`boolean`	`removeShutdownHook(Thread hook)` De-registers a previously-registered virtual-machine shutdown hook.	boolean	removeShutdownHook(Thread hook) Unregisters a previously registered virtual machine shutdown hook.
`void`	`runFinalization()` Runs the finalization methods of any objects pending finalization.	void	runFinalization() Provides a hint to the virtual machine that it would be useful to attempt to perform any outstanding object finalization.
`static void`	`runFinalizersOnExit(boolean value)` **Deprecated.** *This method is inherently unsafe. It may result in finalizers being called on live objects while other threads are concurrently manipulating those objects, resulting in erratic behavior or deadlock.*	static void	runFinalizersOnExit(boolean run) *This method is deprecated. This method is unsafe.*
`long`	`totalMemory()` Returns the total amount of memory in the Java virtual machine.	long	totalMemory() Returns the total amount of memory which is available to the running program.
`void`	`traceInstructions(boolean on)` Enables/Disables tracing of instructions.	void	traceInstructions(boolean enable) Switches the output of debug information for instructions on or off.

Exhibit E

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
void `traceMethodCalls(boolean on)` Enables/Disables tracing of method calls.	void traceMethodCalls(boolean enable) Switche s the output of debug information for methods on or off.
Methods inherited from class java.lang.Object `clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`	**Inherited Methods[1]** ▶From class java.lang.Object
	Inherited Methods[2] ▼From class java.lang.Object

Object	clone() Creates and returns a copy of this `Object`.
boolean	equals(Object o) Compares this instance with the specified object and indicates if they are equal.
void	finalize() Called before the object's memory is reclaimed by the VM.
final Class<? extends Object>	getClass() Returns the unique instance of `Class` that represents this object's class.

[1] Collapsed view.

Exhibit E

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
	int hashCode() Returns an integer hash code for this object.
	final void notify() Causes a thread which is waiting on this object's monitor (by means of calling one of the `wait()` methods) to be woken up.
	final void notifyAll() Causes all threads which are waiting on this object's monitor (by means of calling one of the `wait()` methods) to be woken up.
	String toString() Returns a string containing a concise, human-readable description of this object.
	final void wait() Causes the calling thread to wait until another thread calls the `notify()` or `notifyAll()` method of this object.
	final void wait(long millis, int nanos) Causes the calling thread to wait until another thread calls the `notify()` or `notifyAll()` method of this object or until the specified timeout expires.

[2] Expanded view.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
	final void wait(long millis) Causes the calling thread to wait until another thread calls the `notify()` or `notifyAll()` method of this object or until the specified timeout expires.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
Method Detail	**Public Methods**
addShutdownHook `public void addShutdownHook(Thread hook)` Registers a new virtual-machine shutdown hook. The Java virtual machine *shuts down* in response to two kinds of events: • The program *exits* normally, when the last non-daemon thread exits or when the `exit` (equivalently, `System.exit`) method is invoked, or • The virtual machine is *terminated* in response to a user interrupt, such as typing `^C`, or a system-wide event, such as user logoff or system shutdown. A *shutdown hook* is simply an initialized but unstarted thread. When the virtual machine begins its shutdown sequence it will start all registered shutdown hooks in some unspecified order and let them run concurrently. When all the hooks have finished it will then run all uninvoked finalizers if finalization-on-exit has been enabled. Finally, the virtual machine will halt. Note that daemon threads will continue to run during the shutdown sequence, as will non-daemon threads if shutdown was initiated by invoking the `exit` method. Once the shutdown sequence has begun it can be stopped only by invoking the `halt` method, which forcibly terminates the virtual machine.	public void **addShutdownHook** (Thread hook) Since: API Level 1 Registers a virtual-machine shutdown hook. A shutdown hook is a `Thread` that is ready to run, but has not yet been started. All registered shutdown hooks will be executed once the virtual machine shuts down properly. A proper shutdown happens when either the `exit(int)` method is called or the surrounding system decides to terminate the application, for example in response to a `CTRL-C` or a system-wide shutdown. A termination of the virtual machine due to the `halt(int)` method, an `Error` or a `SIGKILL`, in contrast, is not considered a proper shutdown. In these cases the shutdown hooks will not be run. Shutdown hooks are run concurrently and in an unspecified order. Hooks failing due to an unhandled exception are not a problem, but the stack trace might be printed to the console. Once initiated, the whole shutdown process can only be terminated by calling `halt()`. If `runFinalizersOnExit(boolean)` has been called with a `true` argument, garbage collection and finalization will take place after all hooks are either finished or have failed. Then the virtual machine terminates. It is recommended that shutdown hooks do not do any time-consuming activities, in order to not hold up the shutdown process longer than necessary. **Parameters** *hook* the shutdown hook to register. **Throws**

Exhibit E

Java^TM 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
Once the shutdown sequence has begun it is impossible to register a new shutdown hook or de-register a previously-registered hook. Attempting either of these operations will cause an `IllegalStateException` to be thrown. Shutdown hooks run at a delicate time in the life cycle of a virtual machine and should therefore be coded defensively. They should, in particular, be written to be thread-safe and to avoid deadlocks insofar as possible. They should also not rely blindly upon services that may have registered their own shutdown hooks and therefore may themselves in the process of shutting down. Shutdown hooks should also finish their work quickly. When a program invokes `exit` the expectation is that the virtual machine will promptly shut down and exit. When the virtual machine is terminated due to user logoff or system shutdown the underlying operating system may only allow a fixed amount of time in which to shut down and exit. It is therefore inadvisable to attempt any user interaction or to perform a long-running computation in a shutdown hook. Uncaught exceptions are handled in shutdown hooks just as in any other thread, by invoking the `uncaughtException` method of the thread's `ThreadGroup` object. The default implementation of this method prints the exception's stack trace to `System.err` and terminates the thread; it does not cause the virtual machine to exit or halt. In rare circumstances the virtual machine may *abort*, that is, stop running without shutting down cleanly. This occurs when the virtual machine is terminated externally, for example with the `SIGKILL` signal on Unix or the `TerminateProcess` call on Microsoft Windows. The virtual machine may also abort if a native method goes awry by, for	*IllegalArgumentException* — if the hook has already been started or if it has already been registered. *IllegalStateException* — if the virtual machine is already shutting down. *SecurityException* — if a SecurityManager is registered and the calling code doesn't have the RuntimePermission("shutdownHooks").

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

example, corrupting internal data structures or attempting to access nonexistent memory. If the virtual machine aborts then no guarantee can be made about whether or not any shutdown hooks will be run.
 	
Parameters:
`hook` - An initialized but unstarted `Thread` object
 	
Throws:
`IllegalArgumentException` - If the specified hook has already been registered, or if it can be determined that the hook is already running or has already been run
`IllegalStateException` - If the virtual machine is already in the process of shutting down
`SecurityException` - If a security manager is present and it denies `RuntimePermission("shutdownHooks")`
Since:
1.3
See Also:
`removeShutdownHook(java.lang.Thread)`, `halt(int)`, `exit(int)`	

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
availableProcessors `public int availableProcessors()` Returns the number of processors available to the Java virtual machine. This value may change during a particular invocation of the virtual machine. Applications that are sensitive to the number of available processors should therefore occasionally poll this property and adjust their resource usage appropriately. **Returns:** the maximum number of processors available to the virtual machine; never smaller than one **Since:** 1.4	public int **availableProcessors** () Since: API Level 1 Returns the number of processors available to the virtual machine. **Returns** the number of available processors, at least 1.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
exit `public void exit(int status)` Terminates the currently running Java virtual machine by initiating its shutdown sequence. This method never returns normally. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination. The virtual machine's shutdown sequence consists of two phases. In the first phase all registered `shutdown hooks`, if any, are started in some unspecified order and allowed to run concurrently until they finish. In the second phase all uninvoked finalizers are run if `finalization-on-exit` has been enabled. Once this is done the virtual machine `halts`. If this method is invoked after the virtual machine has begun its shutdown sequence then if shutdown hooks are being run this method will block indefinitely. If shutdown hooks have already been run and on-exit finalization has been enabled then this method halts the virtual machine with the given status code if the status is nonzero; otherwise, it blocks indefinitely. The `System.exit` method is the conventional and convenient means of invoking this method. **Parameters:** `status` - Termination status. By convention, a nonzero status code indicates abnormal termination. **Throws:** `SecurityException` - If a security manager is present and its `checkExit` method does not permit exiting with the specified status **See Also:**	public Process **exec** (String[] progArray, String[] envp) Since: API Level 1 Executes the specified command and its arguments in a separate native process. The new process uses the environment provided in `envp`. Calling this method is equivalent to calling `exec(progArray, envp, null)`. **Parameters** *progArray* the array containing the program to execute as well as any arguments to the program. *envp* the array containing the environment to start the new process in. **Returns** the new `Process` object that represents the native process. **Throws** *IOException* if the requested program can not be executed. *SecurityException* if the current `SecurityManager` disallows program execution. **See Also** `checkExec(String)` public Process **exec** (String prog, String[] envp, File directory) Since: API Level 1 Executes the specified program in a separate native process. The new process uses the environment provided in `envp` and the working directory

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

`SecurityException`, `SecurityManager.checkExit(int)`, `addShutdownHook(java.lang.Thread)`, `removeShutdownHook(java.lang.Thread)`, `runFinalizersOnExit(boolean)`, `halt(int)`
 	
…
 	
exec
 	
`public Process exec(String command)`
`        throws IOException`
Executes the specified string command in a separate process.
 	
This is a convenience method. An invocation of the form `exec(command)` behaves in exactly the same way as the invocation `exec(command, null, null)`.
 	
Parameters:
`command` - a specified system command.
Returns:
A new `Process` object for managing the subprocess
Throws:
`SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess
`IOException` - If an I/O error occurs
`NullPointerException` - If `command` is `null`
`IllegalArgumentException` - If `command` is empty
See Also:
`exec(String[], String[], File)`, `ProcessBuilder`
 	
exec
	specified by `directory`.
	
	Parameters
	prog the name of the program to execute.
	envp the array containing the environment to start the new process in.
	directory the directory in which to execute the program. If `null`, execute if in the same directory as the parent process.
	
	Returns
	the new `Process` object that represents the native process.
	Throws
	IOException if the requested program can not be executed.
	SecurityException if the current `SecurityManager` disallows program execution.
	
	See Also
	`checkExec(String)`
	
	public Process **exec** (String[] progArray, String[] envp, File directory)
	Since: API Level 1
	
	Executes the specified command and its arguments in a separate native process. The new process uses the environment provided in `envp` and the working directory specified by `directory`.
	
	Parameters

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

`public Process exec(String command, String[] envp) throws IOException`
Executes the specified string command in a separate process with the specified environment.
 	
This is a convenience method. An invocation of the form `exec(command, envp)` behaves in exactly the same way as the invocation `exec(command, envp, null)`.
 	
Parameters:
`command` - a specified system command.
`envp` - array of strings, each element of which has environment variable settings in the format *name=value*, or `null` if the subprocess should inherit the environment of the current process.
Returns:
A new `Process` object for managing the subprocess
Throws:
`SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess
`IOException` - If an I/O error occurs
`NullPointerException` - If `command` is `null`, or one of the elements of `envp` is `null`
`IllegalArgumentException` - If `command` is empty
See Also:
`exec(String[], String[], File)`, `ProcessBuilder`
 	
exec
 	
`public Process exec(String command, String[] envp, File dir)`
	progArray the array containing the program to execute as well as any arguments to the program.
	
	envp the array containing the environment to start the new process in.
	
	directory the directory in which to execute the program. If `null`, execute if in the same directory as the parent process.
	
	Returns
	the new `Process` object that represents the native process.
	Throws
	IOException if the requested program can not be executed.
	SecurityException if the current `SecurityManager` disallows program execution.
	
	See Also
	`checkExec(String)`
	
	public Process **exec** (String prog, String[] envp)
	Since: API Level 1
	
	Executes the specified program in a separate native process. The new process uses the environment provided in `envp`. Calling this method is equivalent to calling `exec(prog, envp, null)`.
	
	Parameters
	prog the name of the program to execute.

Exhibit E

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

`throws IOException`
Executes the specified string command in a separate process with the specified environment and working directory.
 	
This is a convenience method. An invocation of the form `exec(command, envp, dir)` behaves in exactly the same way as the invocation `exec(cmdarray, envp, dir)`, where `cmdarray` is an array of all the tokens in `command`.
 	
More precisely, the `command` string is broken into tokens using a `StringTokenizer` created by the call `new StringTokenizer(command)` with no further modification of the character categories. The tokens produced by the tokenizer are then placed in the new string array `cmdarray`, in the same order.
 	
Parameters:
`command` - a specified system command.
`envp` - array of strings, each element of which has environment variable settings in the format *name*=*value*, or `null` if the subprocess should inherit the environment of the current process.
`dir` - the working directory of the subprocess, or `null` if the subprocess should inherit the working directory of the current process.
Returns:
A new `Process` object for managing the subprocess
Throws:
`SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess
`IOException` - If an I/O error occurs
`NullPointerException` - If `command` is `null`, or one of the elements of `envp` is `null`
`IllegalArgumentException` - If `command` is empty
	envp the array containing the environment to start the new process in.
	
	Returns
	the new `Process` object that represents the native process.
	Throws
	IOException if the requested program can not be executed.
	SecurityException if the current `SecurityManager` disallows program execution.
	
	See Also
	`checkExec(String)`
	
	public Process **exec** (String prog)
	Since: API Level 1
	
	Executes the specified program in a separate native process. The new process inherits the environment of the caller. Calling this method is equivalent to calling `exec(prog, null, null)`.
	
	Parameters
	prog the name of the program to execute.
	
	Returns
	the new `Process` object that represents the native process.
	Throws
	IOException if the requested program can not be executed.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
Since: 1.3 **See Also:** `ProcessBuilder`	*SecurityException* — if the current `SecurityManager` disallows program execution. **See Also** `checkExec(String)`
exec `public Process exec(String[] cmdarray) throws IOException` Executes the specified command and arguments in a separate process. This is a convenience method. An invocation of the form `exec(cmdarray)` behaves in exactly the same way as the invocation `exec(cmdarray, null, null)`. **Parameters:** `cmdarray` - array containing the command to call and its arguments. **Returns:** A new `Process` object for managing the subprocess **Throws:** `SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess `IOException` - If an I/O error occurs `NullPointerException` - If `cmdarray` is `null`, or one of the elements of `cmdarray` is `null` `IndexOutOfBoundsException` - If `cmdarray` is an empty array (has length `0`) **See Also:** `ProcessBuilder`	public Process **exec** (String[] progArray) Since: API Level 1 Executes the specified command and its arguments in a separate native process. The new process inherits the environment of the caller. Calling this method is equivalent to calling `exec(progArray, null, null)`. **Parameters** *progArray* — the array containing the program to execute as well as any arguments to the program. **Returns** the new `Process` object that represents the native process. **Throws** *IOException* — if the requested program can not be executed. *SecurityException* — if the current `SecurityManager` disallows program execution. **See Also** `checkExec(String)`
exec	public void **exit** (int code) Since: API Level 1

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

`public Process exec(String[] cmdarray, String[] envp) throws IOException`
Executes the specified command and arguments in a separate process with the specified environment.
 	
This is a convenience method. An invocation of the form `exec(cmdarray, envp)` behaves in exactly the same way as the invocation `exec(cmdarray, envp, null)`.
 	
Parameters:
`cmdarray` - array containing the command to call and its arguments.
`envp` - array of strings, each element of which has environment variable settings in the format *name*=*value*, or `null` if the subprocess should inherit the environment of the current process.
Returns:
A new `Process` object for managing the subprocess
Throws:
`SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess
`IOException` - If an I/O error occurs
`NullPointerException` - If `cmdarray` is `null`, or one of the elements of `cmdarray` is `null`, or one of the elements of `envp` is `null`
`IndexOutOfBoundsException` - If `cmdarray` is an empty array (has length `0`)
See Also:
`ProcessBuilder`
 	
exec
 	
`public Process exec(String[] cmdarray, String[] envp,`
	Causes the virtual machine to stop running and the program to exit. If `runFinalizersOnExit(boolean)` has been previously invoked with a `true` argument, then all objects will be properly garbage-collected and finalized first.
	
	Parameters
	code the return code. By convention, non-zero return codes indicate abnormal terminations.
	
	Throws
	SecurityException if the current `SecurityManager` does not allow the running thread to terminate the virtual machine.
	
	See Also
	`checkExit(int)`

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

`File dir)`
`throws IOException`
Executes the specified command and arguments in a separate process with the specified environment and working directory.
 	
Given an array of strings `cmdarray`, representing the tokens of a command line, and an array of strings `envp`, representing "environment" variable settings, this method creates a new process in which to execute the specified command.
 	
This method checks that `cmdarray` is a valid operating system command. Which commands are valid is system-dependent, but at the very least the command must be a non-empty list of non-null strings.
 	
If `envp` is `null`, the subprocess inherits the environment settings of the current process.
 	
`ProcessBuilder.start()` is now the preferred way to start a process with a modified environment.
 	
The working directory of the new subprocess is specified by `dir`. If `dir` is `null`, the subprocess inherits the current working directory of the current process.
 	
If a security manager exists, its `checkExec` method is invoked with the first component of the array `cmdarray` as its argument. This may result in a `SecurityException` being thrown.
 	
Starting an operating system process is highly system-dependent. Among the many things that can go wrong are:	

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

• The operating system program file was not found.
• Access to the program file was denied.
• The working directory does not exist.
 	
In such cases an exception will be thrown. The exact nature of the exception is system-dependent, but it will always be a subclass of `IOException`.
 	
Parameters:
`cmdarray` - array containing the command to call and its arguments.
`envp` - array of strings, each element of which has environment variable settings in the format *name*=*value*, or `null` if the subprocess should inherit the environment of the current process.
`dir` - the working directory of the subprocess, or `null` if the subprocess should inherit the working directory of the current process.
Returns:
A new `Process` object for managing the subprocess
Throws:
`SecurityException` - If a security manager exists and its `checkExec` method doesn't allow creation of the subprocess
`IOException` - If an I/O error occurs
`NullPointerException` - If `cmdarray` is `null`, or one of the elements of `cmdarray` is `null`, or one of the elements of `envp` is `null`
`IndexOutOfBoundsException` - If `cmdarray` is an empty array (has length `0`)
Since:
1.3
See Also:
`ProcessBuilder`	

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
freeMemory `public long freeMemory()` Returns the amount of free memory in the Java Virtual Machine. Calling the `gc` method may result in increasing the value returned by `freeMemory`. **Returns:** an approximation to the total amount of memory currently available for future allocated objects, measured in bytes.	public long **freeMemory** () Since: API Level 1 Returns the amount of free memory resources which are available to the running program. **Returns** the approximate amount of free memory, measured in bytes.
gc `public void gc()` Runs the garbage collector. Calling this method suggests that the Java virtual machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the virtual machine has made its best effort to recycle all discarded objects. The name `gc` stands for "garbage collector". The virtual machine performs this recycling process automatically as needed, in a separate thread, even if the `gc` method is not invoked explicitly. The method `System.gc()` is the conventional and convenient means of invoking this method.	public void **gc** () Since: API Level 1 Indicates to the virtual machine that it would be a good time to run the garbage collector. Note that this is a hint only. There is no guarantee that the garbage collector will actually be run.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

getLocalizedInputStream
 	
@Deprecated
`public InputStream` **`getLocalizedInputStream`**`(InputStream in)`
Deprecated. *As of JDK 1.1, the preferred way to translate a byte stream in the local encoding into a character stream in Unicode is via the `InputStreamReader` and `BufferedReader` classes.*
Creates a localized version of an input stream. This method takes an `InputStream` and returns an `InputStream` equivalent to the argument in all respects except that it is localized: as characters in the local character set are read from the stream, they are automatically converted from the local character set to Unicode.
 	
If the argument is already a localized stream, it may be returned as the result.
 	
Parameters:
`in` - InputStream to localize
Returns:
a localized input stream
See Also:
`InputStream`, `BufferedReader.BufferedReader(java.io.Reader)`, `InputStreamReader.InputStreamReader(java.io.InputStream)`
	public InputStream **getLocalizedInputStream** (InputStream stream)
	Since: API Level 1
	This method is deprecated.
	Use `InputStreamReader`.
	Returns the localized version of the specified input stream. The input stream that is returned automatically converts all characters from the local character set to Unicode after reading them from the underlying stream.
	
	Parameters
	stream the input stream to localize.
	
	Returns
	the localized input stream.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

getLocalizedOutputStream
 	
@Deprecated
`public OutputStream getLocalizedOutputStream(OutputStream out)`
Deprecated. *As of JDK 1.1, the preferred way to translate a Unicode character stream into a byte stream in the local encoding is via the `OutputStreamWriter`, `BufferedWriter`, and `PrintWriter` classes.*
Creates a localized version of an output stream. This method takes an `OutputStream` and returns an `OutputStream` equivalent to the argument in all respects except that it is localized: as Unicode characters are written to the stream, they are automatically converted to the local character set.
 	
If the argument is already a localized stream, it may be returned as the result.
 	
Parameters:
`out` - OutputStream to localize
Returns:
a localized output stream
See Also:
`OutputStream`, `BufferedWriter.BufferedWriter(java.io.Writer)`, `OutputStreamWriter.OutputStreamWriter(java.io.OutputStream)`, `PrintWriter.PrintWriter(java.io.OutputStream)`
	public OutputStream **getLocalizedOutputStream** (OutputStream stream)
	Since: API Level 1
	
	This method is deprecated.
	Use `OutputStreamWriter`.
	
	Returns the localized version of the specified output stream. The output stream that is returned automatically converts all characters from Unicode to the local character set before writing them to the underlying stream.
	
	Parameters
	stream the output stream to localize.
	
	Returns
	the localized output stream.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
getRuntime `public static Runtime getRuntime()` Returns the runtime object associated with the current Java application. Most of the methods of class `Runtime` are instance methods and must be invoked with respect to the current runtime object. **Returns:** the `Runtime` object associated with the current Java application.	public static Runtime **getRuntime** () Since: API Level 1 Returns the single `Runtime` instance. **Returns** the `Runtime` object for the current application.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

halt
 	
`public void halt(int status)`
Forcibly terminates the currently running Java virtual machine. This method never returns normally.
 	
This method should be used with extreme caution. Unlike the `exit` method, this method does not cause shutdown hooks to be started and does not run uninvoked finalizers if finalization-on-exit has been enabled. If the shutdown sequence has already been initiated then this method does not wait for any running shutdown hooks or finalizers to finish their work.
 	
Parameters:
`status` - Termination status. By convention, a nonzero status code indicates abnormal termination. If the `exit` (equivalently, `System.exit`) method has already been invoked then this status code will override the status code passed to that method.
Throws:
`SecurityException` - If a security manager is present and its `checkExit` method does not permit an exit with the specified status
Since:
1.3
See Also:
`exit(int)`, `addShutdownHook(java.lang.Thread)`, `removeShutdownHook(java.lang.Thread)`
	public void **halt** (int code)
	Since: API Level 1
	
	Causes the virtual machine to stop running, and the program to exit. Neither shutdown hooks nor finalizers are run before.
	
	Parameters
	code the return code. By convention, non-zero return codes indicate abnormal terminations.
	
	Throws
	SecurityException if the current `SecurityManager` does not allow the running thread to terminate the virtual machine.
	
	See Also
	`checkExit(int)`
	`addShutdownHook(Thread)`
	`removeShutdownHook(Thread)`
	`runFinalizersOnExit(boolean)`

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

load
 	
`public void load(String filename)`
Loads the specified filename as a dynamic library. The filename argument must be a complete path name. From `java_g` it will automagically insert "_g" before the ".so" (for example `Runtime.getRuntime().load("/home/avh/lib/libX11.so");`).
 	
First, if there is a security manager, its `checkLink` method is called with the `filename` as its argument. This may result in a security exception.
 	
This is similar to the method `loadLibrary(String)`, but it accepts a general file name as an argument rather than just a library name, allowing any file of native code to be loaded.
 	
The method `System.load(String)` is the conventional and convenient means of invoking this method.
 	
Parameters:
`filename` - the file to load.
Throws:
`SecurityException` - if a security manager exists and its `checkLink` method doesn't allow loading of the specified dynamic library
`UnsatisfiedLinkError` - if the file does not exist.
`NullPointerException` - if `filename` is `null`
See Also:
`getRuntime()`, `SecurityException`, `SecurityManager.checkLink(java.lang.String)`
	public void **load** (String pathName)
	Since: API Level 1
	
	Loads and links the dynamic library that is identified through the specified path. This method is similar to `loadLibrary(String)`, but it accepts a full path specification whereas `loadLibrary` just accepts the name of the library to load.
	
	Parameters
	pathName the absolute (platform dependent) path to the library to load.
	
	Throws
	UnsatisfiedLinkError if the library can not be loaded.
	SecurityException if the current `SecurityManager` does not allow to load the library.
	
	See Also
	`checkLink(String)`

Exhibit E

Java™ 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

loadLibrary
 	
`public void loadLibrary(String libname)`
Loads the dynamic library with the specified library name. A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent. The mapping from a library name to a specific filename is done in a system-specific manner.
 	
First, if there is a security manager, its `checkLink` method is called with the `libname` as its argument. This may result in a security exception.
 	
The method `System.loadLibrary(String)` is the conventional and convenient means of invoking this method. If native methods are to be used in the implementation of a class, a standard strategy is to put the native code in a library file (call it `LibFile`) and then to put a static initializer:
 	
`static { System.loadLibrary("LibFile"); }`
 	
within the class declaration. When the class is loaded and initialized, the necessary native code implementation for the native methods will then be loaded as well.
 	
If this method is called more than once with the same library name, the second and subsequent calls are ignored.
 	
Parameters:
`libname` - the name of the library.
Throws:
	public void **loadLibrary** (String libName)
	Since: API Level 1
	
	Loads and links the library with the specified name. The mapping of the specified library name to the full path for loading the library is implementation-dependent.
	
	Parameters
	libName the name of the library to load.
	
	Throws
	UnsatisfiedLinkError if the library can not be loaded.
	SecurityException if the current `SecurityManager` does not allow to load the library.
	
	See Also
	`checkLink(String)`

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

`SecurityException` - if a security manager exists and its `checkLink` method doesn't allow loading of the specified dynamic library
`UnsatisfiedLinkError` - if the library does not exist.
`NullPointerException` - if `libname` is `null`
See Also:
`SecurityException`, `SecurityManager.checkLink(java.lang.String)`	

maxMemory `public long maxMemory()` Returns the maximum amount of memory that the Java virtual machine will attempt to use. If there is no inherent limit then the value `Long.MAX_VALUE` will be returned. **Returns:** the maximum amount of memory that the virtual machine will attempt to use, measured in bytes **Since:** 1.4	public long **maxMemory** () Since: API Level 1 Returns the maximum amount of memory that may be used by the virtual machine, or `Long.MAX_VALUE` if there is no such limit. **Returns** the maximum amount of memory that the virtual machine will try to allocate, measured in bytes.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

removeShutdownHook
 	
`public boolean` **`removeShutdownHook`**`(Thread hook)`
De-registers a previously-registered virtual-machine shutdown hook.
Parameters:
`hook` - the hook to remove
Returns:
`true` if the specified hook had previously been registered and was successfully de-registered, `false` otherwise.
Throws:
`IllegalStateException` - If the virtual machine is already in the process of shutting down
`SecurityException` - If a security manager is present and it denies `RuntimePermission("shutdownHooks")`
Since:
1.3
See Also:
`addShutdownHook(java.lang.Thread)`, `exit(int)`
	public boolean **removeShutdownHook** (Thread hook)
	Since: API Level 1
	
	Unregisters a previously registered virtual machine shutdown hook.
	
	Parameters
	hook the shutdown hook to remove.
	
	Returns
	`true` if the hook has been removed successfully; `false` otherwise.
	Throws
	IllegalStateException if the virtual machine is already shutting down.
	
	SecurityException if a SecurityManager is registered and the calling code doesn't have the RuntimePermission("shutdownHooks").

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

runFinalization
 	
`public void runFinalization()`
Runs the finalization methods of any objects pending finalization. Calling this method suggests that the Java virtual machine expend effort toward running the `finalize` methods of objects that have been found to be discarded but whose `finalize` methods have not yet been run. When control returns from the method call, the virtual machine has made a best effort to complete all outstanding finalizations.
 	
The virtual machine performs the finalization process automatically as needed, in a separate thread, if the `runFinalization` method is not invoked explicitly.
 	
The method `System.runFinalization()` is the conventional and convenient means of invoking this method.
 	
See Also:
`Object.finalize()`
	public void **runFinalization** ()
	Since: API Level 1
	
	Provides a hint to the virtual machine that it would be useful to attempt to perform any outstanding object finalization.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)

runFinalizersOnExit
 	
@Deprecated
`public static void `**`runFinalizersOnExit`**`(boolean value)`
Deprecated. *This method is inherently unsafe. It may result in finalizers being called on live objects while other threads are concurrently manipulating those objects, resulting in erratic behavior or deadlock.*
Enable or disable finalization on exit; doing so specifies that the finalizers of all objects that have finalizers that have not yet been automatically invoked are to be run before the Java runtime exits. By default, finalization on exit is disabled.
 	
If there is a security manager, its `checkExit` method is first called with 0 as its argument to ensure the exit is allowed. This could result in a SecurityException.
 	
Parameters:
`value` - true to enable finalization on exit, false to disable
Throws:
`SecurityException` - if a security manager exists and its `checkExit` method doesn't allow the exit.
Since:
JDK1.1
See Also:
`exit(int)`, `gc()`, `SecurityManager.checkExit(int)`
	public static void **runFinalizersOnExit** (boolean run)
	Since: API Level 1
	
	This method is deprecated.
	This method is unsafe.
	
	Sets the flag that indicates whether all objects are finalized when the virtual machine is about to exit. Note that all finalization which occurs when the system is exiting is performed after all running threads have been terminated.
	
	Parameters
	run `true` to enable finalization on exit, `false` to disable it.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
totalMemory `public long totalMemory()` Returns the total amount of memory in the Java virtual machine. The value returned by this method may vary over time, depending on the host environment. Note that the amount of memory required to hold an object of any given type may be implementation-dependent. **Returns:** the total amount of memory currently available for current and future objects, measured in bytes.	public long **totalMemory** () Since: API Level 1 Returns the total amount of memory which is available to the running program. **Returns** the total amount of memory, measured in bytes.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
traceInstructions `public void traceInstructions(boolean on)` Enables/Disables tracing of instructions. If the `boolean` argument is `true`, this method suggests that the Java virtual machine emit debugging information for each instruction in the virtual machine as it is executed. The format of this information, and the file or other output stream to which it is emitted, depends on the host environment. The virtual machine may ignore this request if it does not support this feature. The destination of the trace output is system dependent. If the `boolean` argument is `false`, this method causes the virtual machine to stop performing the detailed instruction trace it is performing. **Parameters:** `on` - `true` to enable instruction tracing; `false` to disable this feature.	public void **traceInstructions** (boolean enable) Since: API Level 1 Switches the output of debug information for instructions on or off. On Android, this method does nothing. **Parameters** *enable* `true` to switch tracing on, `false` to switch it off.

Exhibit E

Java^{TM} 2 Platform Standard Edition 5.0 API Specification (java.lang.Runtime)	Android APIs (java.lang.Runtime)
traceMethodCalls `public void traceMethodCalls(boolean on)` Enables/Disables tracing of method calls. If the `boolean` argument is `true`, this method suggests that the Java virtual machine emit debugging information for each method in the virtual machine as it is called. The format of this information, and the file or other output stream to which it is emitted, depends on the host environment. The virtual machine may ignore this request if it does not support this feature. Calling this method with argument false suggests that the virtual machine cease emitting per-call debugging information. **Parameters:** `on` - `true` to enable instruction tracing; `false` to disable this feature.	public void **traceMethodCalls** (boolean enable) Since: API Level 1 Switches the output of debug information for methods on or off. **Parameters** *enable* `true` to switch tracing on, `false` to switch it off.